APPENDIX A

Patent Application for

LOW OVERHEAD METHODS AND APPARATUS FOR SHARED ACCESS STORAGE DEVICES

SANergy FSMDC Specification

Version 2.2.5

6 July 2000

Author & Technical Contact:
Chris Stakutis, CTO SANergy
SANergy, Westford MA
Tivoli Storage Business Unit
617 693 9770
cstakutis@tivoli.com

SECURITY NOTICE:

This document contains information of a proprietary nature. ALL INFORMATION CONTAINED HEREIN SHOULD BE KEPT IN CONFIDENCE. This information should not be divulged to persons other than Tivoli employees authorized by the nature of their duties to receive such information, or individuals and organizations authorized by Tivoli Storage Business Unit in accordance with existing policy regarding the release of company information.

About This Document

This document is the Specifications for the **FSMDC** layer of the SANergy SAN-based file system sharing product.

Who Uses This Document

3rd party customers and partners
Strategy & Architecture
Technical Office(s)
Designers
Developers
Function Test
System Test
Performance
Legal Counsel
National Language Technical Center (NLTC)
Other Tivoli Laboratories

Trademarks

The following names have been adopted by the International Business Machines Corporation as trademarks in the United States and/or other countries and maybe used throughout this document:

IBM
Tivoli
S/390
AIX
DB2

Windows, Windows NT, Windows 2000 are registered trademarks of Microsoft Corporation.

Solaris is a registered trademark of Sun Microsystems.

HP-UX is a registered trademark of Hewlett-Packard Company.

Acronyms and Terminology

Acronyms are used to streamline text in this document. Acronyms are not always defined after their first usage, however. Reasons for this include: many of the acronyms are commonly used and known by the intended audience of this document, the purpose of this document is primarily reference, and a glossary is provided to define new and less commonly used acronyms and terms.

Types of Notes

The following types of notes are used in this document:

Note:

> Either explains aspects of a definition that may not be obvious to the reader or provides auxiliary information about it.

Design Note:

> Design questions/notes/issues are contained in boxes as is this text.

Table Of Contents

Section 1. SANergy System Overview

1.1 Background

SANergy software enables multiple host computers to directly attach to storage devices and safely share file data. Without SANergy (or similar software), the machines would each believe that all the storage was theirs and only theirs, and thus not coordinate each other's caches, overwrite allocations, mark or otherwise write to disks when unauthorized, and rollback (undo) transactions being performed by another machine. SANergy solves those problems plus allows heterogeneous systems to share data. The fundamental approach of SANergy is to capitalize on traditional LAN networking for the majority of the management of "sharing" activity (metadata), but use the SAN fabric for the actual file data I/O.

SANergy does not introduce a new file system to the world. Rather, it takes existing file systems and adds some layers of software to provide the correct behavior in a shared-SAN file system environment. Each file system in the SAN has a single machine that is the true file system owner, referred to as a Meta Data Controller (MDC). The other participants are referred to as SANergy clients (of *that* file system). To date, SANergy only supports a small number of file systems and MDC platforms. The purpose of this document is to fully describe the functionality that is need by a particular file system in order for SANergy to consider using it as a SANergy-supported one.

Note:

At various times and in various documents, the terms "volume" and "file system" have been used interchangeably. Either term is correct and there is no implied distinction. Other terms, such as "disk", are very specific and always mean a "physical amount of storage that appears to the host computer as a disk" (and **never** means or implies a file system or volume).

1.2 MDC's and Clients

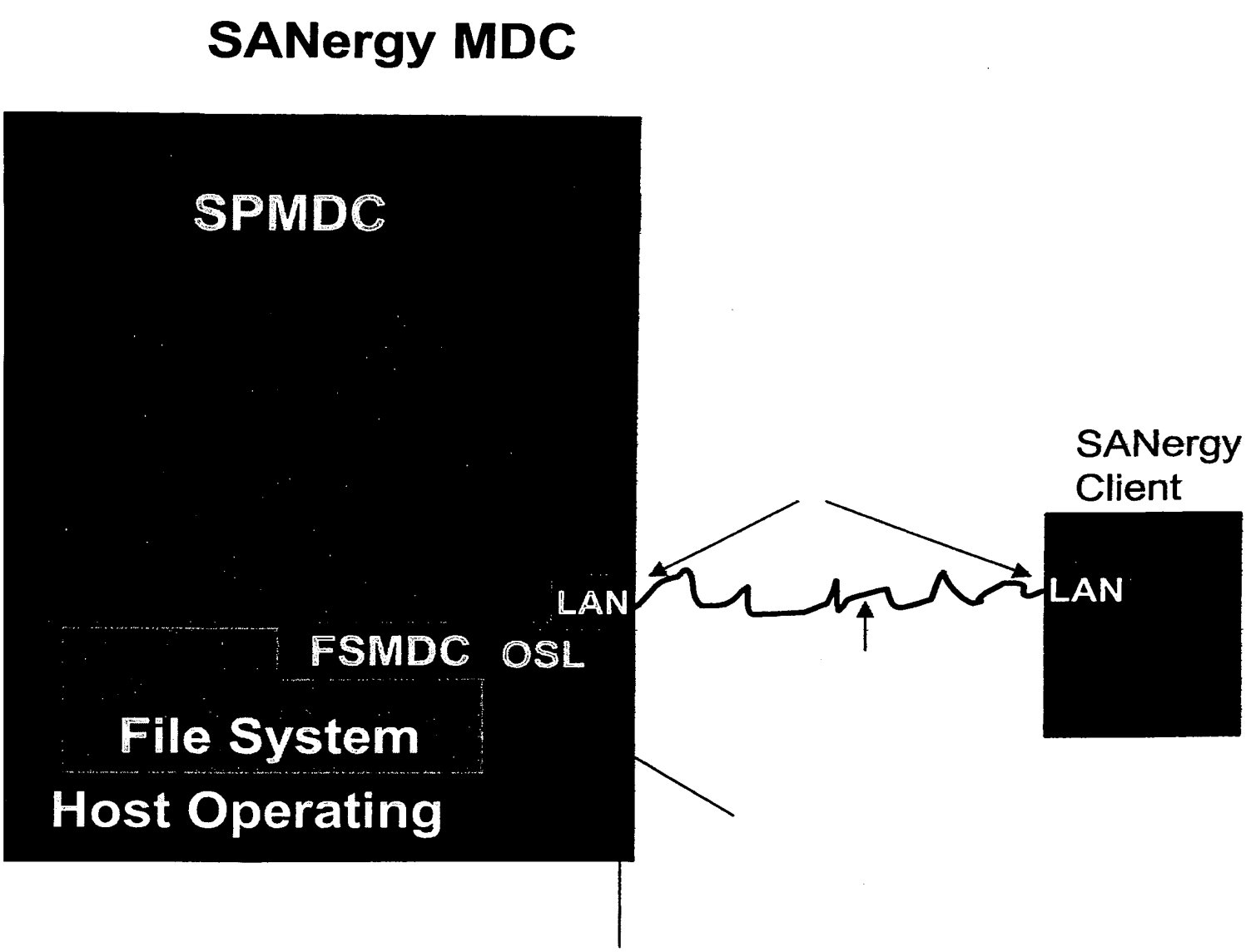
A SANergy system consists of two distinct types of machines playing in the game: MDC's (metadata controllers) and Client systems. The terms MDC and Client are a per-logical-file-system basis. That is, a physical machine could be an MDC for one or two volumes, and a Client to the rest, or any other combination (there is only a single MDC per volume but no limit to the number of clients).

The MDC machine is the only machine that hard-mounts (read-write) the volume and lays down a file system and manages that file system with respect to authentication, security, administration, and allocation. The philosophy of SANergy is to exploit existing operating system platforms and their file systems by making them work correctly in a SAN data sharing world. In one respect, an MDC is somewhat like a file-server for a volume; other machines use conventional networking to access the volume and thus are authenticated and controlled in the traditional networking model. However, only meta-data (file open, security, allocation information, etc.) take place in the SANergy approach; the actual file payload (data) is transferred directly between the storage elements and the accessing computers.

The Client machines use ordinary networking to mount the volumes and open files. The SANergy code on the Client intercepts reads and writes and issues those directly to the disk elements. It is able to do this because it has coordinated with the MDC (regarding caching) and asked the MDC for the list of physical blocks for the file (aka "the map").

Determining the block list and coordinating the access to the blocks is very file system and platform specific. SANergy supports a few different MDC platforms directly, and the goal of this document is to allow other file systems and platforms to build interfaces that work correctly in a SANergy environment.

1.3 Architecture Overwiew



File system providers need to provide the functionality of the box labeled "FSMDC". This document details that exact API including the rules and responsibilities as well as the programming interfaces. The SANergy Protocol (SP) describes the LAN interface between an MDC and a client (data structures and functionality). Depending on the host platform and various business arrangements, the main box (SPMDC) could be built by the SANergy team or by the File system and/or operating system vendor.

There are separate documents for all the other functional layers. This document details the FSMDC layer only.

1.4 References - Applicable Documents

There are a number of applicable documents for creating full MDC.

- OSL Document: Describes the Operating Specific Layers and services needed by higher levels of the MDC.
- SP Document: Describes the SANergy Protocol that goes across the LAN wire. All the functions, rules, and obligations are also described.
- LAN Document: Describes base LAN layer functionality.

Section 2. FSMDC Overview

2.0 Introduction

There are two basic capabilities that the FSMDC must provide:

- Provide "volume-to-disk" translations.
- Provide file disk map information.

Ultimately, SANergy clients access a file's data by reading and writing directly to physical storage. Thus, a SANergy client must be told the exact physical information regarding the file. This is done in a two-tier hierarchy: a physical description of the physical disks that comprise a logical volume and then the description of the file blocks relative to that volume information.

There are many details regarding how long such transmitted description information is valid and what steps are necessary to protect the overall "system" integrity of the data (i.e. caches).

SANergy makes an attempt to simplify the overall task of allowing multiple computers to share physical storage directly. In fact, the patented approach is novel enough that most people admire the overall simplicity. However, it is often over simplified in the SANergy marketing and presentation literature. There are quite a few rules and responsibilities of a properly coded MDC and those regulations must be strictly watched. The most important rule is regarding proper flushing and purging of MDC caches. File systems in general have always had an ability to "flush" dirty data to storage, but there is a new and more important need with a SAN system: purging. Purging means removing pages from the MDC memory system because those corresponding pages on disk might have been written by another machine.

A properly written FSMDC will provide for 100% safe, reliable, coherent, and recoverable byte-level data sharing between multiply attached hosts.

2.1 Packaging and Co-Existence

"fsmdc.h" describes the functions, parameters, and codes necessary for this interface layer. These functions must be provided by the file system manufacturer as the functions are very file system specific. They need to be wrapped-up into a single shared-object library which can be dynamically loaded by the higher level SANergy MDC code at runtime. It is entirely possible that on a single MDC there may be more than one file system present and thus more than one of these libraries. The layers above the FSMDC are designed to exhaustively load all detected FSMDC modules (shared libraries), this way new file system support can occur independent of SANergy releases.

2.2 Names, Cookies, and Locks

2.2.1 Names

All functions throughout the SANergy layers that accept names for files, volumes, shares, or anything else, use the NT-style UTF-16 UNICODE encoding. It is imperative that this requirement be recognized by the various file system FSMDC providers as SANergy lives in a multinational world. If a given file system does not support UNICODE names internally, it must do the best job possible to translate the incoming names appropriately.

2.2.2 Cookies

The FSMDC layer introduces a concept of a "file system specific and unique identifier" called *cookies* used for referencing both Files and Volumes. It is possible for a file while in use to have its name changed or even deleted. It is also possible for multiple separate files to share a common name. The use of cookies avoids any confusion.

Before higher layers of SANergy can manipulate a file (extend the map, set a file size, etc.), it must first ask for a Cookie for that file (by name). All subsequent operations on a given "session" of a file will use that cookie and never the name again. Any and all reports from the FSMDC layer regarding a file (such as the allocation map) will also report back the associated volume cookie. Volumes can also change in definition while the file system is up and running (e.g. a volume can be extended, striping changed, or other physical disk members added). Any change to a volume MUST result in a different cookie for the volume. Thus, dynamically, SANergy will be able to detect any material changes to the file system and files (even while files are in-use).

2.2.3 Locks and Leases

One of the most important functions provided by the FSMDC layer is the delivery of file-map information (the physical description of all the blocks that comprise a given file). This information has a life-span associated with it ("lease"). The SPMDC layer decides how long a map is considered useable and reports back that information along with the map to the caller. The caller (SANergy client) is obligated to ***never*** reference any part of the map or the blocks it represents beyond the lease duration.

While a lease is active the file's allocated blocks are considered "locked". The FSMDC guarantees that there will be no removal or reclamation of any of those blocks of the file (note, the current file-size is a separate concept and somewhat unrelated to whether blocks associated with a file are still associated with the file). An "unlock()" function is provided to allow the higher layers to inform the FSMDC layer that the file map is no longer being referenced and thus the FSMDC layer is now free to reclaim any blocks or completely reorganize a file (perhaps defragging). If the lease on the file expires it is *exactly* as if the a call to "unlock()" was made.

The FSMDC does **not** need to keep "counters" regarding how many times a particular file was locked. A single "unlock()" will unlock the file. The higher levels of software (SPMDC) will track how many parties are active on a file and issue or defer issuing the unlock() call.

The SANergy client will provide a desired lease value to the SPMDC layer. It is typical for SANergy to ask for very short leases (i.e. 5 seconds) for files opened for read-only access. Files opened for read-write access SANergy is likely to request a much longer lease. To save overhead, SANergy clients will typically let leases expire rather than specifically "close" the session (particularly for read-only files).

To review, the FSMDC layer concerns itself with delivering file maps and considers the allocated blocks of a file "locked" until the unlock() function is called. The SPMDC layer has a concept of leases and manages calling the unlock() function as needed.

2.2.4 File Size and Extended Size

With SANergy, a file has two size values: the current reported file length (File Size) and the Allocation Size. For efficiency, when creating or extending a file, SANergy will ask that the file grow by a very large amount (100MB for instance). It is assumed that the cost of allocating a small amount of space is about the same (overall) as allocating a larger amount, and minimizing the number of times a file is extended is extremely important (LAN transmissions and related issues).

A file's size may be changed at any time to any value by the local MDC or any SANergy clients. The Allocation size, however, is only allowed to *grow*, until there are no locked-maps active. That is, if the MDC provides a locked-map to a requester, it is obligated to never reclaim and reassign those blocks until the file is unlocked (regardless if the request comes from a local process on the MDC or a SANergy client).

The MDC should report "file size" to any inquiring application, and *not* the allocation size. The SANergy GetMap() function allows the MDC to report both values. Similarly, there are functions provided by the FSMDC layer to allow SANergy change either or both values.

2.2.5 Volumes

SANergy uses a two level hierarchy for fully describing where all the disks blocks are that comprise a given file. A file-map in the SANergy sense is a map of blocks relative to a prior disclosed volume. A volume is an aggregate of physical disks, offsets and pieces of those disks, and possibly an algorithm (e.g. NT RAID-0 striping) over the disks. A volume is referred to as a "glom" in fsmdc documentation which implies that there is some algorithm applied to the set of disks when interpreting the file-map information (typically some striping algorithm or spanning).

It is not required for any SANergy client system to run the same volume manager as the MDC. To be exact, the clients go below any volume manager anyway to access the disks. This is to allow the widest range of computers to connect and successfully run in a SANergy environment.

Using the two layer abstraction allows for far more compact map descriptions. Without a volume abstraction, describing a large file that is striped across several drives could result in a map that is many times larger than in this approach.

SANergy is designed to work correctly in any disk environment and does not count on world-wide-names of devices (that Fibre Channel allows). Thus, for a client system to locate a particular disk that the MDC references, it is sometimes necessary to perform a search. The FSMDC must describe to the clients a unique identifier somewhere on each disk (such as a disk signature). The API provides for the MDC informing the clients exactly what the offset of the identifier is and how many bytes long it is. Clients will quickly scan the disks and typically build a table of this information.

3.0 API's

This chapter will provide details of each and every API the FSMDC layer must provide. It will sometimes show particular data structures and function parameters, however, this document might be rev'd differently from the actual "fsmdc.h" header file. The header file is the definitive definition and this chapter shall serve as a guide and offer supporting verbiage (but might be slightly inaccurate over time).

Each FSMDC implementation is required to provide all of the functions described in this section and the header file. The layers above the FSMDC will definitely be looking-up and calling each of these functions.

3.1 Data types

The "fsmdc.h" file defines many key and portable datatypes. The bulk of the FSMDC functions and definitions do not use such data types as "long" or "long long" or even "byte" or "char". All such data types have an "FS" equivalent that is designed to provide a consistent definition across all platforms (for example, a "long" on Tru-64 is 64 bits).

The actual definitions are now in a file called "fstypes.h" which is 'included' by fsmdc.h.

3.2 FS_GetVersion()

This is typically the first function to be called by a given MDC implementation. It will inform the SPMDC layer of what generation this particular FSMDC is from. Recall, the FSMDC modules can be distributed by parties that are not directly from SANergy and could thus have various ages to them.

This function returns a FSLONG value indicating the non-decimal hundred's version value. That is, version 2.20 would be reported back as "220" in decimal.

3.3 FS_GetVolCookie()

This function requests a volume "cookie" given a pathname. The pathname could be something that merely represents the disk (e.g. "c:\" or "/exports/test1") or could embody a full file name with it. The goal is to decode the volume that is referenced by the parameter and return a cookie for it (so that a subsequent call can get details about the make-up of the volume if desired).

This function might not be called frequently. It is conceivable that the caller might first call the "GetMap" function which returns a volume cookie directly.

3.4 FS_GetVolByCookie()

Given a volume cookie (either from a GetMap() or GetVolCookie() request) return the detailed information about the make-up of the volume.

The caller provides a data area for this function to write its description. It is entirely possible that the caller does not provide enough space and hence specific return value indicating this. The return value of 'msglen' will indicate the total message length (amount of data written into the buffer) or the amount of data that *would* be written into the buffer if the buffer was large enough (thus, in a condition where the caller buffer is not large enough, the caller can look at this value to determine to correct size to allocate).

This message as well as others also provide a "vendorStatus" field. Every FSMDC function is required to return an offical FS error code. Many times the FSMDC supplier has more information available to it in an

error condition and this field allows the FS to disclose an FS-specific value. In the case of an error, the disks data structure can contain an UNICODE error string that details the error.

This function reports back an "array" of descriptions of the disks that comprise this particular file system, as well as particular information regarding how these disks should be treated as a volume-set. The 'GlomType" field might indicate a certain type of striping, or spanning, or something else. If a particular FSMDC has a new type of volume information, it could always merely report back the set of disks, and then later in the GetMap requests report individual (verbose-physical) descriptions for each block. In any case, this report must list all disks that could be possibly referenced by a subsequent GetMap request. The caller (a SANergy client) will seek out and locate each physical disk after making this call; if it can't find all of the (non "META") disks referenced, it will not consider this volume useable by SANergy.

- Special Note on "META" disks: The FSMDC has an ability to tell higher layers about disks that are logically part of a volume but contain no file data and those wont be ever referenced in a file-map. These are "META" disks and identified by the special attribute bit in the disk-info structure. SANergy clients should not expect to find such disks on the SAN and it is there not considered to be an error if these disks do not exist. Some file systems completely separate out file-data from file-meta-data and store all the meta data on physically separate disks (for performance), yet these "META" disks are still considered part of the overall volume. Those meta disks could be local/internal disks on the MDC, and thus not visible on the SAN.

The 'BlockSize' field is used by the SANergy clients to determine the smallest possible I/O that can correctly be issued to the storage device. Typically this value is 512.

Disk Details

The FSDISK structure details the information necessary for SANergy clients to locate and use the disk correctly. Up to 16 bytes of "unique" data is available for the FSMDC to specify a given disk (DiskID). This ID can live at any arbitrary byte offset on the physical device. An additional value (???) indicates just how long this particular value is for this FSMDC.

The BlockOffset field states where the (block wise) on the device this particular partition/slice starts. The subsequent file maps will be relative to this value, and not to the physical block-0 of the device. The nBlocks field indicates how blocks beyond the "BlockOffset" one could possibly be referenced. SANergy will consider it an error if a map ever references a block beyond this range.

3.5 FS_GetCookies()

Before a file-map can be requested for a file, a set of cookies for that file must be learned. Recall, cookies are unique and are persistent across deletes or renamings of files or volumes (that is, they are never reused). This function accepts a full pathname and reports back the associated volume and file cookies.

If the caller is not familiar with the volume cookie (possibly because it is the first time this volume is referenced, or possibly this is the first time since this volume has been extended or otherwise altered), it will then call the GetVolByCookie() function learn about the details.

3.6 FS_GetLockedMap()

The main work-horse function of an MDC is the GetMap function (actually called "FS_GetLockedMap()" but often referred to as simply GetMap()).

Maps can be potentially quite large, plus, a file might not be fully resident (in the case of an HSM). The caller requests for a range of information regarding a file, and this function may return that range or a sub-set of that range (providing for HSM issues).

As with the GetVol function, the caller specifies an output area to hold the variable-length map description. It is possible that the caller has not provided enough space and thus there is a specific error message indicating this.

The caller can specify a set of behavioral flags. The "WAIT" flag tells the FSMDC that the caller wants the entire requested range and will not tolerate a partial map. This is sometimes needed in the case of an HSM situation; the caller might have already used the initially reported map and now needs the whole map regardless of how long it takes. The "NO_HOLE" flag says that the caller expects that the return map contain all useable blocks and that "holes" are not permitted. In such a case, the FSMDC must first create real storage for such areas.

There is flexibility in describing a file map. The FSMDC must choose the most compact method that correctly represents the file (the difference could be several thousand bytes different and where this information is typically transmitted over LAN, it is critical to use the most compact method available).

3.6.1 Lock

When this function is called, the FSMDC is required to "lock" the allocated range of a file, which means that under no circumstances should the file's allocation size be permitted to shrink (even if a local application on the FSMDC truncates or removes the file). The blocks must remain assigned to this file until the subsequent FS_UnLock() function is called for this file.

3.6.2 Partial and sub-maps

It is possible for this function to return only a sub-range of the map-range requested (return code FS_M_SUBMAP). This is to allow for file systems were it can take a great deal of time to build the entire extent list and it is assumed that the caller can start consuming the file with a partial map. This is also useful with HSM products where some small portion of a file might be resident, but the rest needs to be fetched from backing store. The caller can re-call this API to ask for the remaining range and possibly specify the FS_WAIT flag. This is not to be confused with the return code of FS_M_PARTIAL which means that the caller did not allocate enough space to hold the entire information.

3.6.2.1 Glom Or Physical

File maps are returned as a list of "extents". There are a number of different extent-types defined in the fsmdc.h file. As mentioned above, the MDC is required to return the map in the most compact type possible (for instance, an ordinary file that has no holes and no special regions and is described in its entirety can use the very compact FSEXTENT_COMPACT_GLOM type. The difference in size of the various extent data structure choices is quite large (more than double) and thus the total map transmitted could be quite different size-wise.

Most of the extent types are glom-relative (and with the _GLOM identifier). This means that the client side uses specific knowledge (i.e. about striping and striping algorithms) to infer the physical disks that underlay this extent. One particular extent type, FSEXTENT_VERBOSE_PHYSICAL is not glom-relative and instead each extent references a specific disk. This is a catch-all to allow clients that do not understand certain MDC algorithms to still be able to get useable map information (at the expense of the map being larger). A client can specifically ask for this extent type by specifying the FS_M_FLAG_VERBOSE_PHYSICAL flag on the request.

3.6.2.2 Disk Ordinals

Some of the extent types have a "disk ordinal" value associated with each extent. This field has various meanings based on the overall map-type returned. For "compact", there is no such field present as the glom type is sufficiently straightforward that the client will know which disk member(s) underlay this extent. For the _SIMPLE_GLOM and _VERBOSE_GLOM types this value will indicate a glom-specific "start" value (if needed, and some map-types such as FS_GLOM_SIMPLE wont need it). For the _VERBOSE_PHYSICAL extent types, this value is the disk that this particular extent underlays and there is no implicit tie-in to other disks).

3.6.3 Validity of a block

The 'validity' field of the verbose map indicates whether a client can directly access a particular extent. The "DIRECT_ACCESS" flag indicates that it is safe for a client to go directly to storage for this data (which is the typical case). Other values should not be directly accessed. The "HOLE" flag indicates that this extent represent a logical "hole" in the file. In such a case, the client can simulate zeros for "read" operations. For "write" operations, the client must either re-ask for the map and specify the "NO_HOLE" flag or send the "write" over the conventional technology (LAN). For "COMPRESSED" or "ENCRYPTED" the client will also send the request over the conventional technology (unless it knows for sure what the encryption or compression technology is).

3.6.4 Purging

Whenever this function is called, the FSMDC layer is **required** to both flush and purge any data related to this file from its buffer caches. It is CRITICAL that the FSMDC purge the blocks such that any subsequent local references to the file data result in the data being retrieved from disk. If this requirement is not followed, then the file will become incoherent and applications will receive the wrong contents.

3.6.5 Byte Range Locks

The FSMDC is responsible for honoring file locks (byte range lock) requests from local and network applications. SANergy clients will issue ordinary network byte-range-lock requests.

It is **required** that the FSMDC flush and purge the corresponding lock region any time a byte-range-lock request is made and the file is currently locked (meaning that a map has been served). If it does not do this, corruption will result.

3.6.6 Forced Verbose Maps

The caller can specify the FS_M_FLAG_VERBOSE_PHYSICAL bit in the flags parameter which means that the caller insists on receiving back a map in the verbose-physical format. It is critical that this flag be supported. The reason is that not all SANergy clients will have an inherent ability to understand all glom types. New glom types will come along, and/or older SANergy clients will be in the field and might not understand certain glom types. When a SANergy client receives a map that has a glom type it does not support, it will re-ask for the map with this bit set indicating that it will accept the lengthy verbose *physical* description instead (which every client is required to handle).

3.7 FS_UnlockMap()

This simple function merely unlocks the specified file. After this function completes, the FSMDC is free to reallocate, reassign, or reclaim any of the blocks of this file. There will be only a single UnlockMap() issued for a file regardless of how many "GetLockedMap()" calls have been made.

3.8 FS_SetFileSizes()

Any file that is open for writing may indeed need to change either the current file size or the allocated size or both. For efficiency, SANergy clients will over-allocate space for a file so that it does not have to ask for space for each and every write operation. This is called "hyper allocation". When the file is unlocked (via FS_UnlockMap()), the FSMDC is free to reclaim the over-allocated space (until a call to Unlock is made, the FSMDC promises to not reorganize or deallocate the blocks that comprise the file).

This function is used to both inform the FSMDC of a file's current file size as it is growing, as well as to increase the over-allocation space. It will never cause a reduction in the allocated space as that can only happen by the FSMDC itself, and only once the file is in an unlocked state.

When this function is called specifying a larger allocation size than is current (an "extend" operation), the FSMDC must allocate the amount specified and ensure that there is "real storage" behind the allocations (thus "not sparce"). If it cannot perform the allocation (as in a disk-full situation) it must attempt to acquire as much as possible. The response will indicate if the new allocation size succeeded and the client-side will interpret any shortage as a disk-full situation. SANergy Clients routinely "hyper extend"; that is, over allocate a file by a great many megabytes while the file is growing.

When the MDC is requested to extend a file, it should not zero-out the newly assigned blocks. The client side owns this responsibility (and this leads to much higher performance).

Any time this API is successfully executed, the file's "modification date" needs to be set to the current time. This allows clients that are streaming/growing a file to have an ability to effect the date, which other clients may wish to observe (particularly important for backup and HSM applications).

Specifying file size as negative one means to **not** change or set the file size at all (but still have the by-product behavior of considering the file changed as of the moment of this call, for reporting and inquiry purposes).

Specifying AllocSized to –1 indicates that no change is desired to the allocation size (presumably there is a non-negative-1 FileSize specified).

Attempting to decrease the allocation quantity is benign; allocations will only be discarded once a file map is unlocked. File-size can be set to any value between negative-1 and the current allocation size (greater than the allocation size is an error).

The 'flags' value is for future use, possibly to describe to the FS what type of blocks to allocate or how contiguous or similar.

The 'minalloc' size tells the FSMDC layer the absolute minimum size needed and if that can't be allocated then return an error. In general, the caller wants the 'allocsize' allocated, but in a near disk full situation might be willing to settle for less.

Files that are multiply opened for interleaved append operations pose special difficulty. It is possible for each client to have a different opinion of the current file size. Although the resultant interleaving of data might be somewhat different than in a non-SANergy case, the file should still be "correct" (that is, there should be no uninitialized data and no extraneous data). Even with SANergy, network caching and flushing can result in odd interleaving. Thus, if a 100% correctly interleaved data is desired, the applications should be using byte-range locks to correctly coordinate their activity.

APPENDIX B

Patent Application for

LOW OVERHEAD METHODS AND APPARATUS FOR SHARED ACCESS STORAGE DEVICES

SANergy SP Specification

Version 2.2.3

22 June 2000

Author & Technical Contact:
Chris Stakutis, CTO SANergy
SANergy, Westford MA
Tivoli Storage Business Unit
617 693 9770
cstakutis@tivoli.com

NOTE:

The hardcopy version of this document is FOR REFERENCE ONLY.

It is the responsibility of the user to ensure that this is the current version. Any outdated hardcopy is invalid and must be removed from possible use. It is also the responsibility of the user to ensure the completeness of this document prior to use.

SECURITY NOTICE:

This document contains information of a proprietary nature. ALL INFORMATION CONTAINED HEREIN SHOULD BE KEPT IN CONFIDENCE. This information should not be divulged to persons other than Tivoli employees authorized by the nature of their duties to receive such information, or individuals and organizations authorized by Tivoli Storage Business Unit in accordance with existing policy regarding the release of company information.

About This Document

This document is the Specifications for the **FSMDC** layer of the SANergy SAN-based file system sharing product.

Who Uses This Document

3rd party customers and partners
Strategy & Architecture
Technical Office(s)
Designers
Developers
Function Test
System Test
Performance
Legal Counsel
National Language Technical Center (NLTC)
Other Tivoli Laboratories

Trademarks

The following names have been adopted by the International Business Machines Corporation as trademarks in the United States and/or other countries and maybe used throughout this document:

IBM
Tivoli
S/390
AIX
DB2

Windows, Windows NT, Windows 2000 are registered trademarks of Microsoft Corporation.

Solaris is a registered trademark of Sun Microsystems.

HP-UX is a registered trademark of Hewlett-Packard Company.

Acronyms and Terminology

Acronyms are used to streamline text in this document. Acronyms are not always defined after their first usage, however. Reasons for this include: many of the acronyms are commonly used and known by the intended audience of this document, the purpose of this document is primarily reference, and a glossary is provided to define new and less commonly used acronyms and terms.

Types of Notes

The following types of notes are used in this document:

Note:

Either explains aspects of a definition that may not be obvious to the reader or provides auxiliary information about it.

Design Note:

Design questions/notes/issues are contained in boxes as is this text.

Table Of Contents

Section 1. SANergy SP Overview

1.1 Background

The **SANergy** protocol (SP) is the language used by any two SANergy-enabled computers to communicate and coordinate activities. For review, recall that the FSMDC API is the layer of software on an MDC that sits closest to the proprietary file system. The FSMDC layer provides a common interface for generalized SANergy-related MDC file system services. The layer above that provides the true SANergy-related MDC "services" (the activities an MDC must do in order to be a true MDC, which it will use the functions in the lower FSMDC layer to achieve these activities). The MDC provides those services ultimately to SANergy client systems. This document describes the RPC details as well as additional rules and responsibilities of those services.

Note: The reader must first read and fully understand the FSMDC spec before attempting to understand this layer. The implementer of an MDC must adhere to all the rules and responsibilities outlined in the FSMDC document.

The messages between the SANergy client and MDC system are currently one direction only; from client to MDC (with a response, always). The MDC never sends an unsolicited message to a client. Each and every LAN transmission has an encoded "revision" number in the message. This allows the two sides to be a different revision levels and to deal with that accordingly. It is imperative that both sides check and respect the version numbers.

This document, as with the FSMDC one, avoids showing exact data structure or API definitions. The accompanying header file (sp.h) is the definitive source for the details. The purpose of this document is to provide sufficient verbiage and explanations to correctly implement the SP protocol.

1.2 Transports

There are currently two methods of transporting the messages between the client and MDC: the out-of-band IP datagram, and the ioctl() technique. The SP protocol is rich enough to support both of these transports, although the information inside the messages might vary slightly based on the transport in use.

Datagram

Datagram messages are sent to port XXXX (defined in the sp.h header file) of the MDC, and the ephemeral port is used to carry the reply. Datagrams are chosen for their speed and simplicity. The MDC-side should be designed for processing many requests in parallel from many different SANergy client systems.

Ioctl()

The CIFS protocol supports an ability to send a file-system control message from client to server side (sometimes referred to as an ioctl(), or as an fscontrol()). SANergy uses ioctl number XXXX for its messages. A SANergy client that detects it is running through the CIFS protocol will first attempt the ioctl() interface. Should that fail, it will then use the Datagram interface and remember not to bother with ioctl attempts when targeting that server for future requests.

Section 2. Major Concepts

It is assumed that the reader of this document is intimately familiar with the concepts and services provided by the FSMDC layer. The SP protocol essentially bundles those functions up with additional functionality that overall embody what an MDC has to do to be an MDC.

2.1 Minimal Messages

It is a fundamental requirement to minimize the number and size of LAN transactions between SANergy client and server. Thus, some of the messages may encompass several distinct functions that are known to be needed together (and thus reducing the number of transactions). This occasionally leads to parameters that are over-loaded (multiple interpretations) or have different meanings based on the presence of other parameters.

2.2 Leases

Leases are a general SANergy MDC-to-client concept that provide a great deal of operational efficiency. Simply, a lease is a way to request access to a file map for a period of time, and when the period expires, the lease is automatically dissolved (thus, there is no need for an additional expensive message to declare termination of the need for the map). SANergy clients request a lease duration, but the MDC dictates the lease period granted.

The majority of file activities in a system are "open for read" which are then sequentially processed and closed relatively soon after being opened. With that in mind, this layer distinguishes between files opened for reading and those for writing (writing being more complicated). Very short leases are typical for read-only parties (5 or 10 seconds) whereas open-for-write parties have to be tracked typically for longer periods.

Clients are required to respect the lease values. That is, it is illegal to reference a file map (move data to or from disk based on the map) after a lease expires. When a lease is expired, the MDC is free to completely reorganize a file (provided no other leases are in process for that file). Typically, an MDC will immediately clean up any hyper-allocations of the file. At expiration time, the last known file size is considered the high-water mark of the file and the MDC is free to remove blocks between that value and the end of the allocation area. Thus, a client must be careful to apprise the MDC of any growth to a file's size ***before*** the lease on the file expires.

2.3 Tracking

The MDC needs to keep track of file maps that have been served out (and locked). It is entirely possible for a given file to be multiply "opened". The FSMDC layer does not keep counts or stacks related to how many file maps have been served. Thus, it is the responsibility of the MDC to call "UnlockMap()" only when all parties are finished with the file map (hence, it must reference count the parties to a file). This is a design and implementation issue for the MDC layer and not germane to this document, but worth mentioning at this time because the SP protocol introduces a concept of "keys" (described next) for this reason.

2.4 Keys

Each time a SANergy client starts transactions to a file (and thus receives a map for the file), the client becomes registered as a party to the file and a "key" is generated. The key is then used by the client for subsequent transactions to that file. Keys are similar to file-handles; that is, there is one-per-active-session on the file. The key values themselves are unique from a given MDC (but not unique across the SANergy system). The MDC side generates the keys and then uses them internally to track each map dealt out (for the purpose of cleaning up expired maps or looking up references from the client).

2.5 'future' fields

Nearly every LAN data structure in this document has a 'future' field which is typically 8 unsigned long words long. The purpose is to make it simpler to add capabilities or make small changes to the protocol without having to re-do the data structure. 'future' fields are guaranteed to be zero (the sender must zero-out the data structure prior to sending).

2.6 Message header and overview

Every message between the client and MDC (in either direction) has the same overall format, described by this data structure:

- *Note:* Any data structure or function definitions presented in this document are to be used for annotation and discussion purposes only and should be considered the official definitions. The official definitions are found in the appropriate header files (e.g. sp.h or lan.h).

```
typedef struct {
            FSULONG endian;
        FSULONG revision;
        char    messageID[16];
        FSLONG  functionCode;
        FSULONG length;
        FSULONG flags;
        FSLONG  result;
        FS_SYSTEM system;
} LAN_MsgHeader;
```

The above structure is the preamble for each and every message between SANergy members. The SP layer (sp.h) defines messages all of which have this data structure as the first element.

The endian field indicates the endianness of *this* message. The observer of the message will know whether or not binary data values need to be endian-adjusted by comparing the value in this field against the endian constant. If they match, no adjustment is necessary. Each and every message sent from one computer to another will be layed-down in the natural endianness of the *sending* machine. The receiving machine owns the responsibility of adjusting the data if necessary. Even "replies" to messages will be in the endianness of the machine *sending* the reply.

The 'revision' field is the critical link between the two sides so that they can be sure to speak the same dialect of this API (or reject communication if desired). As of this writing, the current version number is 220 (for 2.20) and is referred to by the constant SP_LANVERSION.

The "messageID" field is an arbitrary string that the client creates and that the MDC must use in the response (likely it is some sort of sequence information). The client ensures that each message sent to an MDC will use a unique message ID. Thus, the MDC, if it sees the same message ID twice, it can respond with the exact same previous response. Similarly, if the MDC hasn't yet responded, it will interpret the duplicate as an indication of a client's impatient ness. There is no guarantee, however, of message ID's being globally unique (that is, multiple clients might use the same algorithm/counter to generate the Ids). Clients can't tell ahead of time how long it will take an MDC to process a get-map (or any) request (the time could vary from a millisecond up to several minutes in the case of an HSM fetch). Thus, it is conceivable for a client to ask for a map multiple times, worrying that the MDC may have dropped or ignored the earlier requests.

The 'functionCode' describes the function/service that is being requested and dictates the layout of the data beyond this header in a packet.

The 'length' field indicates the overall amount of data transferred in this message, including this header

and its associated data.

The result field is only interesting on a reply and indicates the overall status of the request. Result values live in "fserr.h".

The 'system' field is a structure that describes as much about the computer system sending the message as possible (e.g. operating system, hardware platform, file system syntax for path separators, and other information). This structure needs to be filled-in on both the sending and replying operations so that the other side can consider behaving differently based on who it is talking with.

Responses

The response back from the MDC to client will be the same overall format (revision, ID, function, length, and then an arbitrary size body). The function code for a response is **always** the negative of the incoming message function code. There are unique responses structures defined for each and every function and are located in the header file below each outbound structure.

2.7 Names

As with the FSMDC layer, any and ALL "name" (or string) type data fields are encoded in the NT style UTF-16 format (never ASCIZ). All such strings will be terminated with a 16-bit zero value (thus the length of the string can be learned by scanning).

Section 3. The Functions

This section will discuss each and every API that can be issued from the client to the MDC (there are currently only five functions). The reader should match up the descriptions in this document against the sp.h header file (this document avoids showing specific definitions as those could slightly change and not effect the overall descriptions in this document.

3.1 LANGetVolume

SANergy clients will need to know the specifics of a volume (file system) regarding physical disks and offsets and sizes and so forth. That information ultimate comes from the MDC's FSMDC layer and there is an SP interface for it, with some differences in the parameters.

The SP_LANGetVolume structure is used to request this information. The 'functionCode' field will be set to the constant SP_LANGETVOLUME. The response structure is SP_LANGetVolumeResp.

A SANergy client possibly needs to be authenticated (license-wise) by an MDC before the MDC should provide services to it. It is convenient to do that authentication at the time of requesting volume information (because volume information must be requested before any useful other SANergy activity can take place). The 'license' field contains the client-side SANergy software license string. The MDC is required to validate that this license is valid, not duplicated (if that is to be prohibited), and so forth. The MDC can reject providing volume information with the FS_E_LICENSE error code. If an MDC does not know how to validate or track licenses, it can return "success".

If the pathname provided is zero-length (either no data or contains a zero-length Unicode string), then the 'cookie' value is considered significant (and this will call GetVolByCookie()). The 'pathname' for this function is the export name or the share, but **not** a true file name or mount point. The implementer of this function will need to translate that share/export name into a real mount point so that the GetVolCookie function can be called correctly. If the translation could not succeed, this function should return one of the SHARE_NAME error values. If this command is activated through the IOCTL interface, then the pathname and cookie fields can be omitted as it is implied by the file object that it is riding on (typically, then, *some* file will have to first be opened, possibly a directory or dummy file on the volume, or similar).

Category Error return values:

- Share name translation (FS_E_SHARExxx)
- License errors (FS_E_LICENSExxx)
- FSMDC errors (from the call to GetVolCookie or GetVolByCookie)
- Misc errors

3.2 LANGetInitialMap

The SANergy client will need an ability to get a physical map for a given file. There are different modes or reasons why a SANergy client will need that information and there are a couple of LAN interfaces created optimized for the various needs.

A new concept is introduced: "intention". The purpose of "intention" is for the client to help the MDC optimize various management tasks by informing the MDC of whether the client will do strictly and only "reads" of a file versus other random activities (writing and stretching a file).

A SANergy client will need maps at various times in a file's "open" life span. The information that the client side has available and its needs are different accordingly. The LANGetInitialMap function is tuned to the needs of a file just being opened by a client for the first time this session (versus a lease expiration or an extend).

The client should call LANClose when it knows that it no longer needs access to map and thus the MDC can start any cleanup operation. Typically this is only needed when a lease period granted is substantially long. For short leases, the kind that are typically granted for a READONLY open for instance, it is more efficient to let the lease just expire (saving an additional LAN transmission).

The data structure SP_LANGetInitialMap defines the input to this function and the response structure is SP_LANGetInitialMapResp. Refer to those definitions in the header file while reading the rest of this section.

The pathname is a two-part UNICODE string (the first part being the share or export name, and the second piece being the relative filename). The first part is null terminated (in UNICODE) and the second part follows immediately (this facilitates having single variable length field). The pathname can be omitted in the case of using the IOCTL interface as the file-object used for the communication implies the specific file.

'filesize' and 'allocsize' can be set to desired initial values or –1 meaning that no change to the existing value is desired. Files opened for some sort of writing would typically like to insist that there are blocks immediately allocated for the file (to save having to request blocks later). Similarly, files opened for "truncate" (or "create new") would like to initially force the file's size to be set to 0 instead of having to issue a separate set-file-size request.

To save LAN transmission, the overall pathname field is considered variable length (thus the last parameter in the structure).

The response typically is a file map in addition to a lease-granted value, the various cookies, and a "key" value. The implementer of this function generates a unique key value and the caller will use that key value for all subsequent communications regarding this *session* of the file. The key is considered "valid" only for the period of the lease. It is not legal to use a key value beyond the lease expiration.

Error return values:

- Results from FS_SetSizes or FS_GetLockedMap()
- Possible out-of-memory conditions.
- File system not supported

3.3 LANGetExtendedMap

While a client is writing new data to a file, it may run out of space from the initial allocation map. The LANGetExtendedMap function requests additional space for the file. The SP_LANGetExtendedMap structure (and ...Resp) is used for this function. The implementation is primarily built on top of the FS_SetSizes() and FS_GetLockedMap() API's. It is unlikely to ever need to stretch a file and not immediately need the new map, which is why this single function will do both operations.

The majority of the parameters for this function are identical to the LANGetInitialMap one. The 'key' is the key value returned by the LANGetInitialMap function (if this file's previous lease has expired, the caller must start over again by calling the LANGetInitialMap function). The remaining parameters are used for calling FS_SetSizes() and FS_GetLockedMap() and have the same rules and responsibilities.

The response is a new map, a new lease value, and a result code.

Error return values:

- Results from FS_SetSizes or FS_GetLockedMap()
- Possible out-of-memory conditions.
- FS_E_KEY_INV : The specified key is unknown, invalid, or possibly expired.

3.4 LANSetSize

It may be desired for a client to merely inform the MDC of a file size change (due to growth while streaming or some other adjustment via a set-end-of-file type of request). As long as the new file size is within the currently "leased" map's allocation area, there is no need for the client to receive new map information (thus more efficient on the MDC and LAN bandwidth). The LANSetSize API (with the SP_LANSetSize data structure) is for this purpose.

The parameters are straightforward. The 'key' value is the value returned from a call to LANGetInitialMap. The 'filesize' value is the new desired file size. The response is even simpler: nothing but a return code value which will hold the value returned by calling FS_SetSizes().

Error return values:

- FS_E_KEY_INV : The specified key is unknown, invalid, or possibly expired.
- Or any value from FS_SetSizes().
- Possibly a "misc" error if the size specified is considered illegal (maybe it is beyond even the allocated size).

3.5 LANClose

A client can express its intention to no longer reference a file (and thus its map). The value of doing this is to allow the MDC to be more efficient and clean up resources associated with tracking leases. For short leases (5 seconds or so) it is typically better to let the lease expire without calling this function (the LAN overhead out weighs the benefit). Longer leases, like those granted for "writes", should call this function. Typically, when a file opened for "write" closes, the final file size needs to be told to the MDC anyway, and so this function allows both "closing" the intention on a file at the same time setting the final file size.

The 'key' value indicates which file is to be closed. Typically this function is called when a file opened for write is closing, and thus it is convenient to be able to specify a final file size (instead of also having to call the LANSetSize function). The 'filesize' parameter can be set to –1 indicating that no change of file size is desired.

Error return values:

- FS_E_KEY_INV : The specified key is unknown, invalid, or possibly expired.
- Or any value from FS_SetSizes().
- Possibly a "misc" error if the size specified is considered illegal (maybe it is beyond even the allocated size).

The client must not use the file key or the file map after calling this function as the MDC is free to tear down all data structures regarding that key.